Pseudorandom number generation on supercomputers

Kleijnen, J.P.C.; Adams, N.

Publication date:
1989

UNIVERSITEIT BRABANT

POSTBOX 90153
5000 LE TILBURG
THE NETHERLANDS

[redacted]

DEPARTMENT OF ECONOMICS

RESEARCH MEMORANDUM

PSEUDORANDOM NUMBER GENERATION ON SUPERCOMPUTERS

Jack P.C. Kleijnen and
Nabil Adams

FEW 378

PSEUDORANDOM NUMBER GENERATION ON SUPERCOMPUTERS

Jack P. C. Kleijnen[1)]

and

Nabil Adam[2)]

February 1989

1) Department of Information Systems and Auditing, School of Business and Economics, Catholic University Brabant (Katholieke Universiteit Brabant), 5000 LE Tilburg, Netherlands. FAX: 013-663019. E-mail: T435KLEI@HTIKUB5.

2) Graduate School of Management, Rutgers University, The State University of New Jersey, 92 New Street, Newark, New Jersey 07102. USA.

PSEUDORANDOM NUMBER GENERATION ON SUPERCOMPUTERS

JACK P.C. KLEIJNEN
Department of Information Systems and Auditing
School of Business and Economics
Catholic University Brabant (Katholieke Universiteit Brabant)
P.O. Box 90153
5000 LE Tilburg, Netherlands

and

NABIL ADAM
Graduate School of Management
Rutgers University
The State University of New Jersey
92 New Street, Newark
New Jersey 07102. USA

Pseudorandom numbers generators are essential in Monte Carlo simulation. On supercomputers these numbers should be generated in parallel. Several procedures are evaluated, and one practical procedure is developed.

C R **Categories and Subject Descriptors:** D. 3.3 [Programming Languages] Procedures, functions and subroutines, G. 1 [Numerical Analysis] Parallel Algorithms, G. 3 [Probability and Statistics] Random Number Generation G. 4 [Mathematical Software] Efficiency, Portability.

General Terms: Algorithms, Experimentation, Performance.

Additional Key Words and Phrases: Simulation, Monte Carlo.

1. INTRODUCTION

Random numbers are the basic elements of stochastic simulation and Monte Carlo models. Examples of such models are simulations of queuing networks and Monte Carlo experiments on location estimators. Such models have been widely used since the advent of computers. In practice, computers do not use truly random numbers. Instead the computer generates pseudorandom numbers, that is, a deterministic algorithm generates outputs which behave as if these outputs were random numbers. This article concentrates on one class of algorithms, namely linear congruential generators. These generators are most popular in management science, mathematical statistics, computer science, and many more scientific disciplines; see Park and Miller (1988). (So Fibonacci and Tausworthe generators are not covered here; see Fishman (1978), Ito and Kanada (1988), Petersen (1988).)

The problem is that, by definition, pseudorandom generators are suspect, that is, deterministic algorithms are surmised to produce non-random outputs. A well-known example is the IBM SYSTEM/360 generator RANDU; see Park and Miller (1988, pp. 1194, 1198). So a particular generator is used only until some researcher develops a practical generator that more closely simulates a truly random number sequence; see Bratley et al. (1983), Fishman (1978), Ripley (1987). Moreover, when new generations of computers are introduced, new generators must be developed. For example, 8-bit personal computers cannot efficiently use algorithms developed for 32-bit machines. More specifically, supercomputers such as the CYBER 205, might employ generators developed for "classical" machines, but such a practice is very inefficient, as we shall show.

2. PSEUDORANDOM GENERATORS ON SUPERCOMPUTERS

Control Data Corporation (CDC) produces the CYBER 200 series computer hardware, which runs FORTRAN 200; see CDC (1986). The FORTRAN 200 language is a superset of standard FORTRAN. This means that standard algorithms can be utilized, but they do not take advantage of the vector or pipeline facilities (these facilities are discussed in the next section). Classical or scalar computers usually employ generators of the linear congruential type:

$$x_{j+1} = (a\, x_j + c) \bmod m \quad (j = 0,1,2,\ldots) \tag{1}$$

where the multiplier a, the constant c, and the modulus m are integers. When c is zero, the generator is called multiplicative congruential. Obviously x_j/m lies between zero and one: $0 \le r_j = x_j/m < 1$. An efficient algorithm results if $m = 2^w$ where w depends on the computer's word size; for example, CDC's Fortran 200 uses $m = 2^{47}$ (see CDC, 1986), whereas IMSL (developed for classical computers) uses $m = 2^{31}-1$. However, there are other considerations than efficiency.

Pseudorandom number generators should yield results r_j, which are statistically independent; that is, the observed sequence $r_o, r_1, \ldots r_n$ should not provide any information about the next sequence $r_{n+1}, r_{n+2}, \ldots$ It turns out to be extremely difficult to meet this requirement, as many authors show; see Bratley et al. (1983), Fishman (1978), Park and Miller (1988), Ripley (1987). It is possible to derive necessary conditions

which, however, are not sufficient. For example, if for computer efficiency a modulus $m = 2^w$ is selected, then a constant c=0 means that a multiplier a=3 (mod 8) yields a cycle length or period (say) $h=m/4= 2^{w-2}$, where a period h means that if the generator starts with "seed" x_0 then $x_h=x_0$ and hence $x_{h+1}= x_1$, and so on. Because these conditions are not sufficient, statistical procedures are applied to the empirical results (r_0, r_1, ...) to test if several types of statistical dependence are absent indeed. For example, two-tuples (r_0, r_1), (r_1, r_2), (r_2, r_3) ... should be uniformly distributed over the unit square.

These considerations explain why in practice users do not specify their own parameters a, c and m; instead they rely on well-tested generators. Examples are the IMSL routine which uses a=16807, c=0, $m=2^{31}-1$ (a=16807 is the default value; two other multipliers are 397204094 and 950706376). In Europe a popular generator is provided by NAG (Numerical Algorithms Group, United Kingdom) with $a=13^{13}$, c=0, and $m=2^{59}$ (double words). Textbooks on simulation discuss other parameter combinations. Let us now return to the CYBER 200 series.

In FORTRAN 200 a scalar function called RANF, is available which uses the multiplicative congruential generator with $m=2^{47}$ and multiplier a=00004C65DA2C866D (hexadecimal); see CDC (1986). To generate a vector of pseudorandom numbers, we can use the subroutine VRANF. Though the documentation on this subroutine is very meager, it is obvious that VRANF does not use the special (pipeline) architecture; that is, VRANF is just a convenient way of programmering a DO-loop that calls the scalar function RANF. For example, generating 50 numbers using RANF and VRANF takes 21 and

35 microseconds while generating 50,000 numbers takes 16,505 and 18,517 microseconds; see An Mey (1983). Next we consider the pipeline architecture in more detail.

3. PIPELINES AND VECTORS

We start with an example, namely the innerproduct of two vectors, $\underset{\sim}{v}_1 \underset{\sim}{v}_2 = \sum_1^n v_{1j} v_{2j}$. This computation requires n scalar operations $v_{1j} v_{2j}$; these n multiplications can be done in parallel because the product $v_{1j} v_{2j}$ does not need the product $v_{1(j-1)} v_{2(j-1)}$. The pipeline architecture of supercomputers means that the computer works as an assembly line; that is, efficiency improves drastically if a large number of identical operations can be executed, independently of each other; see Levine (1982).

All we need to know about pipelining in pseudorandom number generation, is that supercomputers like the CYBER 205 can use their pipelined architecture to improve their efficiency drastically, only if the number of basic operations (such as multiplication) is "large", say $n > 50$, and if these operations can be executed independently or in parallel, which means that recursive statements are not suited to pipelined computers. Unfortunately, the linear congruential generator is recursive: equation (1) shows that to compute x_{j+1} its predecessor x_j is needed.

There is a growing literature on algorithms for vector computers which solve recursion problems in linear algebra (for example, $y_i = x_i - a_i y_{i-1}$ can be solved by so-called recursive doubling). The recursion problem in pseudorandom number generation, however, has gotten less attention. We shall survey several solutions that have been proposed, and present our own solution.

4. PARALLEL PSEUDORANDOM GENERATORS

Suppose that a given simulation experiment requires N pseudorandom numbers in total, for example, N=1,000,000. Number theory gives the period h of a given pseudorandom number generator. The generators that are used in practice, have a relatively long period (for example, $h=2^{30} >> N$) so that, in most cases, the magnitude of N does not need to worry the user.

On a vector computer, however, results should be computed in parallel. In our case, it means that the N pseudorandom numbers should not be computed recursively. Instead (say) J numbers are to be produced in parallel. The literature and manuals suggest that J should to at least 50; otherwise the "assembly line" or vector architecture is inefficient and it is better to use the computer in scalar mode (FORTRAN 200 as a superset of FORTRAN, does permit scalar mode). There is also an upper limit: $J \leq 65,535$ because the CYBER 205 uses 16 bits for addressing; see SARA (1984, p. 26). So the computer should generate J pseudorandom numbers in parallel with $50 < J \leq 65,535$; hence an experiment requiring N numbers, must call this parallel routine [N/J] times where we use [ ] for "rounding upwards to the next integer." For example, if N=1,000,000 and J = 65,535 then 16 calls are necessary. So we may imagine an IxJ matrix of pseudorandom numbers, where J numbers should be generated in parallel and I calls are made to that vector routine. We now survey different solutions to this problem.

5. DIFFERENT MULTIPLIERS

We can generate J pseudorandom numbers in parallel, using J multipliers and constants in the linear congruential relationship; that is equation (1) becomes

$$x_{i+1,j} = a_j x_{ij} + c_j \pmod m$$

$$(j=1,\ldots,J)\ (i=1,2,\ldots,) \qquad (2)$$

So the vector of old numbers $\underset{\sim}{x}_1 = (x_{11}, x_{12}, \ldots, x_{1J})'$ yields the vector of new numbers $\underset{\sim}{x}_2 = (x_{21}, x_{22}, \ldots, x_{2J})'$ or in FORTRAN 200:

```
XNEW (1;J) = A (1;J) * XOLD (1;J) + C(1;J)          (3)
```

where X (1;J) denotes a vector called X, with J elements starting at address 1; see SARA (1984). After the modulus operation, realized in vector mode by the VMOD function, we put

```
XOLD (1;J) = XNEW (1;J)                             (4)
```

We emphasize that the elements <u>within</u> $\underset{\sim}{x}_2$ or XNEW are computed independently (in parallel, in vector mode).

Unfortunately, it is a problem to find J multipliers a_j and constants c_j. We have seen that necessary conditions for the parameters a and c have been derived. These conditions are so weak that, for example, we

can choose from roughly one million multipliers: for $m = 2^{23}$ (half precision on CYBER 205) the following parameters meet the conditions listed in Knuth (1981):

$$a = 2,901 + 8\ k_1 \text{ with } k_1 = 0,1,2,\ldots,1047851 \tag{5}$$

and

$$c = 1,775,001 + 2\ k_2 \text{ with } k_2 = 0,1,2,\ldots,2499 \tag{6}$$

An Mey (1983) proposed sampling a and c from (5) and (6). (We would add that these values should be sampled without replacement; see the next section. Note that for $m = 2^{31}-1$ (a prime number) there are more than 534 million multipliers that yield a full period (h=m-1); see Park and Miller (1988, pp. 1194, 1197). Unfortunately, the conditions (5) and (6) are necessary but not sufficient, so that the statistical behavior of a generator with random parameters is very suspect!

So we prefer to stick to well tested parameters; that is, we prefer existing generators implemented under IMSL, NAG, SIMSCRIPT, and so on. We shall limit the next discussion to multiplicative generators (c_j=0), but our discussion can be extended straightforwardly to $c_j > 0$. So equation (2) becomes

$$x_{i+1,j} = a\ x_{ij} \pmod m$$

$$(j=1,2,\ldots,J)\ (i=1,2,\ldots) \tag{7}$$

6. A VECTOR OF SEEDS

A simple solution is to sample a vector of J seeds (these J seeds are sampled in scalar mode using, for example, RANF in FORTRAN 200). Storing those seeds in XOLD means that equations (3) and (4) become

$$\text{XNEW (1;J)} = A * \text{XOLD (1;J)} \tag{8}$$

$$\text{XOLD (1;J)} = \text{XNEW (1;J)} \tag{9}$$

Unfortunately, such sampling may result in (say) a second seed x_{12} identical to (say) the third value generated in the first columm: $x_{31} = x_{12}$. Such an event means that parts of the "matrix" of numbers are identical ($x_{31} = x_{12}$ implies $x_{41} = x_{22}$, ..., $x_{I1} = x_{(I-2)2}$), and this violates the statistical independence assumption; this assumption is made for the generator and used in the simulation model.

Frederickson et al. (1984) launched a different idea, namely sample the seeds, using a second special generator, say the generator

$$y_{j+1} = b\, y_j + d \pmod m$$

$$j = 0, 1, 2, \ldots \tag{10}$$

This generator is used to sample seeds for the orginal generator of equation (1). Now there are five parameters (a, c, b, d, m) to be selected. Unfortunately, correlations among pseudorandom numbers remain; see Bowman

and Robinson (1987). Moreover, this approach has been worked out for specific generators only; the approach does not cover a generator with (say) $m = 2^{31} - 1$, recommended in Fishman (1978); see Park and Miller (1988) for other recommended parameters. This critique leads to the following idea.

Fishman (1978) proves that, given an initial number or "seed" x_0 and I calls to the "scalar" generator (see equation 1 with c=0), the resulting number x_I can be derived without knowing the intermediate numbers $(x_1, x_2, \ldots, x_{I-1})$:

$$x_I = a^I x_0 \pmod{m}. \tag{10}$$

So if we want to generate J numbers in parallel (such that $I * J \geq N$), then we should start with the following vector of seeds:

$$\underset{\sim}{x}_1 = (x_0, a^I x_0 \pmod{m}, a^{2I} x_0 \pmod{m}, \ldots$$

$$\ldots, a^{(j-1)I} x_0 \pmod{m}, \ldots, a^{(J-1)I} x_0 \pmod{m}) \tag{11}$$

Unfortunately, this mathematical solution cannot be implemented straightforwardly, since overflow occurs when computing $a^{(j-1)I}$. The overflow problem in pseudorandom number computation is also discussed in Park and Miller (1988, p. 1195).

7. A PRACTICAL SOLUTION

We proposed to stick to generators that have been tested extensively, and in which the user has faith. Such a generator may be part of a statistical

package as offered by, for example IMSL and NAG. In the near future, (scalar) generators should be developed that use the full word size of supercomputers; our solution immediately applies to these new generators, as we shall see. We also wish to guarantee the statistical independence of the pseudorandom numbers. This latter condition implies that "streams" (or columm vectors in the I × J matrix formed by x_{ij}) are non-overlapping; that is, these vectors do not contain identical elements.

Fishman (1978, pp. 481-487) has tabulated 400 seeds spaced 100,000 apart, for three different (scalar) multiplicative congruential generators; also see Bratley et al. (1983). These three tables, each with 400 seeds, have been developed in order to decrease the variance of simmulation responses obtained on classical computers. For example, if two queuing systems are to be compared, and one system has the first-in-first-served priority rule while the other system has the smallest-jobs-first rule, then arrival times can be simulated from seed s_1 and service time from seed s_2; if fewer than 100,000 customers arrive then two successive seeds from Fishman's tables suffice; otherwise non-consecutive seeds are used.

We might use Fishman's tabulated values for parallel generation of pseudorandom numbers! Fishman's seeds s_j (j=1, ..., 400) imply $s_2 = x_{100,000} = a\ s_1 \pmod m$ and $s_3 = x_{200,000} = a\ s_2 \pmod m$, and so on. So suppose we use an initial vector of seeds with these 400 seeds: XOLD = $(s_1, s_2, \ldots, s_{400})$; also see equation (8). We can then call the vectorized pseudorandom subroutine 100,000 times before we return to the initial vector. In other words, if the total number of pseudorandom numbers (N) is

smaller than (400 x 100,000 =) 40 million, then this approach yields 400 numbers in parallel. Now we discuss some practical issues and extensions, labeled (i) through (v).

(i) Fishman gives tables only for three specific generators (SIMSCRIPT II, SIMPL/1-LLRANDOM, and $m=2^{31}-1$ with a=397204094). The user may prefer a different generator, for example, NAG's subroutine. We shall solve this problem below.

(ii) Even if one of these three generators is desired, keypunching 400 numbers, each consisting of up to 10 digits, is a slow and error-prone process. Instead we sample one seed from Fishman's table, and have the computer generate the remaining 399 seeds. So the computer uses equation (1) initialized with this particular seed, and after 100,000 calls, the computer stores $s_2 = x_{100,000} = a * x_{99,999} \pmod m$, given specific parameters a and m. In total the computer must generate 399x100,000 pseudorandom numbers! Fortunately, the computer has to do this job only once: all future simulation experiments can use the internally stored table with 400 seeds. Obviously if a particular experiment requires vectors with fewer than 400 elements, then that experiment uses fewer than 400 seeds. Note that storing all 39,900,000 numbers and retrieving them later on, would be impractical: too much space and time would be required.

(iii) If we have the computer generate the seeds to be stored in the initial vector, we are no longer limited to Fishman's three tables! We can then take any generator we like; for example, we can take the generator used in experiments run on a scalar computer; these experiments are

necessary to debug and verify the program that will be run on the supercomputer; also see SARA (1984, p. 5). In the future we can take a generator specifically developed for the supercomputer (see below). Moreover we are no longer limited to a vector length of 400. Sections 3 and 4 showed that the longer the vector is, the more efficient the supercomputer works. So we propose taking the maximum vector length, namely J = 65,535. Then the computer must generate all h different numbers x_j with J = 0, 1, ..., h and known period h; the computer must store 65,535 numbers, namely s_1, s_2, s_3, ..., s_{65535} with $s_2 = x_{I+1}$ and I = [h/65,535]-1, $s_3 = x_{2I+1}$, and so on. So nearly the whole cycle is excecuted (namely from x_0 through $x_{65,534I}$). This initial vector can be generated on a scalar computer or on the supercomputer in scalar mode.

(iv) At the end of a simulation session the user should store the last vector of pseudorandom numbers $\underset{\sim}{x}$ or $\underset{\sim}{r} = \underset{\sim}{x}/m$; all digits need to be saved; see Kleijnen (1986, p. 16). To continue this particular simulation run, the user proceeds from the vector saved at the end of the previous session. If the user wants to execute an unrelated simulation experiment, he or she can take either the last vector or the initial vector provided by the computer center. So on scalar computers the user needs to store a single pseudorandom number; on supercomputers a whole vector must be saved.

(v) Efficiency can be improved by developing generators which take advantage of the wordsize m of a particular supercomputer; that is, new parameter value become relevant for the modulus m. The cycle length h increases as the modulus m increases. For the new modulus, a new value for

the multiplier a needs to be found, applying number theory and mathematical statistics. Finally, a new vector of seeds is to be generated, for the user community. We note that double precision should be avoided on supercomputers since double precision excludes vector mode; SARA (1984, pp. 6,26).

8. CONCLUSIONS

Pseudorandom number generation is a problem that requires the joint efforts of computer scientists for efficient implementation, number theorists for necessary conditions for the generator's parameters, and statisticians for ex post empirical tests. On a supercomputer, the generator should be vectorized in such a way that parallel computation becomes possible. The choice of the generator's parameters is crucial; that is, the sampling of different multipliers yields unacceptable statistical behavior. So the user wishes to stick to well-tested parameter values. Sampling a vector of seeds may result in dependent vectors of pseudorandom numbers. A practical solution is to have a computer generate 65,535 seeds such that independent vectors result. This requires one long run by the computer center, which is an investment to the benefit of all users.

ACKNOWLEDGEMENTS

The first author (Jack Kleijnen) was sponsored by the Supercomputer Visiting Scientist Program at Rutgers University, The State University of New Jersey, during July 1988.

REFERENCES

An Mey, D., Erste Erfahrungen bei der Vektorisiering numerischer Verfahren. (First experiences when vectorizing numerical procedures.) Computer Center, Technical University, Aachen (Germany), July 1983.

Bowman, K.O. and M.T. Robinson, Studies of random number generators for parallel processing. HYPERCUBE MULTIPROCESSOR 1987, edited by M.T. HEATH, SIAM, Philadelphia, 1987, pp. 445-453.

Bratley, P., B. L. Fox and L. E. Schrage, A GUIDE TO SIMULATION, Springer-Verlag, New York, 1983.

CDC,Fortran 200 Version 1 reference manual. Publication no. 60480200, Control Data Corporation, Sunnyvale, California 94088-3492, December 1986.

Fishman, G. S., PRINCIPLES OF DISCRETE EVENT SIMULATION. Wiley-Interscience, New York, 1978.

Frederickson, P., R. Hiromoto, T. L. Jordan, B. Smith and T. Warnock, Pseudorandom trees in Monte Carlo. PARALLEL COMPUTING, 1, 1984, pp. 175-180.

Ito, N. and Y. Kanada, An effective algorithm for the Monte Carlo simulation of the Ising model on a vector processor. SUPERCOMPUTER, May 1988, pp. 31-48.

Kleijnen, J.P.C. Selecting random number seeds in practice. SIMULATION, 47, no. 1, 1986, pp. 15-17.

Knuth, D.E., THE ART OF COMPUTER PROGRAMMING, VOLUME 2. Addison-Wesley, Reading, Massachusetts, 1981.

Levine, R.D., Supercomputers. SCIENTIFIC AMERICAN, January 1982, pp. 112-125.

Park, S.K. and K.W. Miller, Random number generators: good ones are hard to find. COMMUNICATIONS ACM, 31, no. 10, Oct. 1988, pp. 1192-1201.

Petersen, W.P., Some vectorial random number generators for uniform, normal, and Poisson distributions for CRAY X-MP. THE JOURNAL OF SUPERCOMPUTING, 1, 1988, pp. 327-335.

Ripley, B.D., STOCHASTIC SIMULATION. John Wiley & Sons, New York, 1987.

SARA, CYBER 205 USER'S GUIDE; PART. 3, OPTIMIZATION OF FORTRAN PROGRAMS. SARA (Stichting Academisch Rekencentrum Amsterdam, Foundation Academic Computer Center Amsterdam), Amsterdam, November 1984.

* * *

IN 1988 REEDS VERSCHENEN

297 Bert Bettonvil
Factor screening by sequential bifurcation

298 Robert P. Gilles
On perfect competition in an economy with a coalitional structure

299 Willem Selen, Ruud M. Heuts
Capacitated Lot-Size Production Planning in Process Industry

300 J. Kriens, J.Th. van Lieshout
Notes on the Markowitz portfolio selection method

301 Bert Bettonvil, Jack P.C. Kleijnen
Measurement scales and resolution IV designs: a note

302 Theo Nijman, Marno Verbeek
Estimation of time dependent parameters in lineair models
using cross sections, panels or both

303 Raymond H.J.M. Gradus
A differential game between government and firms: a non-cooperative approach

304 Leo W.G. Strijbosch, Ronald J.M.M. Does
Comparison of bias-reducing methods for estimating the parameter in dilution series

305 Drs. W.J. Reijnders, Drs. W.F. Verstappen
Strategische bespiegelingen betreffende het Nederlandse kwaliteits-concept

306 J.P.C. Kleijnen, J. Kriens, H. Timmermans and H. Van den Wildenberg
Regression sampling in statistical auditing

307 Isolde Woittiez, Arie Kapteyn
A Model of Job Choice, Labour Supply and Wages

308 Jack P.C. Kleijnen
Simulation and optimization in production planning: A case study

309 Robert P. Gilles and Pieter H.M. Ruys
Relational constraints in coalition formation

310 Drs. H. Leo Theuns
Determinanten van de vraag naar vakantiereizen: een verkenning van materiële en immateriële factoren

311 Peter M. Kort
Dynamic Firm Behaviour within an Uncertain Environment

312 J.P.C. Blanc
A numerical approach to cyclic-service queueing models

313 Drs. N.J. de Beer, Drs. A.M. van Nunen, Drs. M.O. Nijkamp
Does Morkmon Matter?

314 Th. van de Klundert
Wage differentials and employment in a two-sector model with a dual labour market

315 Aart de Zeeuw, Fons Groot, Cees Withagen
On Credible Optimal Tax Rate Policies

316 Christian B. Mulder
Wage moderating effects of corporatism
Decentralized versus centralized wage setting in a union, firm, government context

317 Jörg Glombowski, Michael Krüger
A short-period Goodwin growth cycle

318 Theo Nijman, Marno Verbeek, Arthur van Soest
The optimal design of rotating panels in a simple analysis of variance model

319 Drs. S.V. Hannema, Drs. P.A.M. Versteijne
De toepassing en toekomst van public private partnership's bij de grote en middelgrote Nederlandse gemeenten

320 Th. van de Klundert
Wage Rigidity, Capital Accumulation and Unemployment in a Small Open Economy

321 M.H.C. Paardekooper
An upper and a lower bound for the distance of a manifold to a nearby point

322 Th. ten Raa, F. van der Ploeg
A statistical approach to the problem of negatives in input-output analysis

323 P. Kooreman
Household Labor Force Participation as a Cooperative Game; an Empirical Model

324 A.B.T.M. van Schaik
Persistent Unemployment and Long Run Growth

325 Dr. F.W.M. Boekema, Drs. L.A.G. Oerlemans
De lokale produktiestructuur doorgelicht.
Bedrijfstakverkenningen ten behoeve van regionaal-economisch onderzoek

326 J.P.C. Kleijnen, J. Kriens, M.C.H.M. Lafleur, J.H.F. Pardoel
Sampling for quality inspection and correction: AOQL performance criteria

327 Theo E. Nijman, Mark F.J. Steel
Exclusion restrictions in instrumental variables equations

328 B.B. van der Genugten
Estimation in linear regression under the presence of heteroskedasticity of a completely unknown form

329 Raymond H.J.M. Gradus
The employment policy of government: to create jobs or to let them create?

330 Hans Kremers, Dolf Talman
Solving the nonlinear complementarity problem with lower and upper bounds

331 Antoon van den Elzen
Interpretation and generalization of the Lemke-Howson algorithm

332 Jack P.C. Kleijnen
Analyzing simulation experiments with common random numbers, part II: Rao's approach

333 Jacek Osiewalski
Posterior and Predictive Densities for Nonlinear Regression.
A Partly Linear Model Case

334 A.H. van den Elzen, A.J.J. Talman
A procedure for finding Nash equilibria in bi-matrix games

335 Arthur van Soest
Minimum wage rates and unemployment in The Netherlands

336 Arthur van Soest, Peter Kooreman, Arie Kapteyn
Coherent specification of demand systems with corner solutions and endogenous regimes

337 Dr. F.W.M. Boekema, Drs. L.A.G. Oerlemans
De lokale produktiestruktuur doorgelicht II. Bedrijfstakverkenningen ten behoeve van regionaal-economisch onderzoek. De zeescheepsnieuwbouwindustrie

338 Gerard J. van den Berg
Search behaviour, transitions to nonparticipation and the duration of unemployment

339 W.J.H. Groenendaal and J.W.A. Vingerhoets
The new cocoa-agreement analysed

340 Drs. F.G. van den Heuvel, Drs. M.P.H. de Vor
Kwantificering van ombuigen en bezuinigen op collectieve uitgaven 1977-1990

341 Pieter J.F.G. Meulendijks
An exercise in welfare economics (III)

342 W.J. Selen and R.M. Heuts
A modified priority index for Günther's lot-sizing heuristic under capacitated single stage production

343 Linda J. Mittermaier, Willem J. Selen, Jeri B. Waggoner, Wallace R. Wood
Accounting estimates as cost inputs to logistics models

344 Remy L. de Jong, Rashid I. Al Layla, Willem J. Selen
Alternative water management scenarios for Saudi Arabia

345 W.J. Selen and R.M. Heuts
Capacitated Single Stage Production Planning with Storage Constraints and Sequence-Dependent Setup Times

346 Peter Kort
The Flexible Accelerator Mechanism in a Financial Adjustment Cost Model

347 W.J. Reijnders en W.F. Verstappen
De toenemende importantie van het verticale marketing systeem

348 P.C. van Batenburg en J. Kriens
E.O.Q.L. - A revised and improved version of A.O.Q.L.

349 Drs. W.P.C. van den Nieuwenhof
Multinationalisatie en coördinatie
De internationale strategie van Nederlandse ondernemingen nader beschouwd

350 K.A. Bubshait, W.J. Selen
Estimation of the relationship between project attributes and the implementation of engineering management tools

351 M.P. Tummers, I. Woittiez
A simultaneous wage and labour supply model with hours restrictions

352 Marco Versteijne
Measuring the effectiveness of advertising in a positioning context with multi dimensional scaling techniques

353 Dr. F. Boekema, Drs. L. Oerlemans
Innovatie en stedelijke economische ontwikkeling

354 J.M. Schumacher
Discrete events: perspectives from system theory

355 F.C. Bussemaker, W.H. Haemers, R. Mathon and H.A. Wilbrink
A (49,16,3,6) strongly regular graph does not exist

356 Drs. J.C. Caanen
Tien jaar inflatieneutrale belastingheffing door middel van vermogensaftrek en voorraadaftrek: een kwantitatieve benadering

357 R.M. Heuts, M. Bronckers
A modified coordinated reorder procedure under aggregate investment and service constraints using optimal policy surfaces

358 B.B. van der Genugten
Linear time-invariant filters of infinite order for non-stationary processes

359 J.C. Engwerda
LQ-problem: the discrete-time time-varying case

360 Shan-Hwei Nienhuys-Cheng
Constraints in binary semantical networks

361 A.B.T.M. van Schaik
Interregional Propagation of Inflationary Shocks

362 F.C. Drost
How to define UMVU

363 Rommert J. Casimir
Infogame users manual
Rev 1.2 December 1988

364 M.H.C. Paardekooper
A quadratically convergent parallel Jacobi-process for diagonal dominant matrices with nondistinct eigenvalues

365 Robert P. Gilles, Pieter H.M. Ruys
Characterization of Economic Agents in Arbitrary Communication Structures

366 Harry H. Tigelaar
Informative sampling in a multivariate linear system disturbed by moving average noise

367 Jörg Glombowski
Cyclical interactions of politics and economics in an abstract capitalist economy

IN 1989 REEDS VERSCHENEN

368 Ed Nijssen, Will Reijnders
"Macht als strategisch en tactisch marketinginstrument binnen de distributieketen"

369 Raymond Gradus
Optimal dynamic taxation with respect to firms

370 Theo Nijman
The optimal choice of controls and pre-experimental observations

371 Robert P. Gilles, Pieter H.M. Ruys
Relational constraints in coalition formation

372 F.A. van der Duyn Schouten, S.G. Vanneste
Analysis and computation of (n,N)-strategies for maintenance of a two-component system

373 Drs. R. Hamers, Drs. P. Verstappen
Het company ranking model: a means for evaluating the competition

374 Rommert J. Casimir
Infogame Final Report

375 Christian B. Mulder
Efficient and inefficient institutional arrangements between governments and trade unions; an explanation of high unemployment, corporatism and union bashing

376 Marno Verbeek
On the estimation of a fixed effects model with selective non-response

377 J. Engwerda
Admissible target paths in economic models